

# Apple II Technical Notes



## Developer Technical Support

### Apple II GS

#### #34: Low-Level QuickDraw II Routines

Revised by: Dave "Evad Snoyl" Lyons, C.K. Haun, Keith Rollin,  
Steven Glass, Matt Deatherage & Eric Soldan

January 1991

Written by: Steven Glass

May 1988

This Technical Note describes the low-level routines which QuickDraw II uses to do much of the work in standard calls and mechanisms for calling these routines and accessing their data.

**Changed since November 1990:** Added a Note on custom bottleneck procedures and updated information on `ShieldCursor` and `UnShieldCursor`.

QuickDraw II lets you customize low-level drawing operations by intercepting the “bottleneck procedures.” QuickDraw II calls an appropriate “bottleneck proc” every time it receives a call to draw an object, measure text, or deal with pictures. For example, if an application calls `PaintOval`, QuickDraw II calls `StdOval` to do the real work, and if an application calls `InvertRgn`, QuickDraw II calls `StdRgn` to do the work.

Installing your own bottleneck procedures is a little bit tricky. The QuickDraw II `SetStdProcs` call accepts a pointer to a 56-byte (\$38 hex) record and fills that record with the addresses of the standard bottleneck procedures of QuickDraw II. You may modify this record by replacing those addresses with the addresses of your own custom bottleneck procedures minus one. (QuickDraw II pushes the address on the stack and executes an RTL to it, so the address in the record must point to the byte before the routine.)

**Note:** A custom bottleneck procedure must not begin at the first byte of a segment. If it does, then the segment could load at the beginning of a bank, and the address minus one would be in the wrong bank and RTL would transfer control to the wrong location. (See Apple II GS Technical Note #90, 65816 Tips and Pitfalls.)

After installing your own procedures, you use `SetGrafProcs` to tell QuickDraw II about them. The format of this call is as follows (taken from the E16.QUICKDRAW file in APW):

```
ostdText      GEQU  $00 ; Pointer - QDProcs -
ostdLine      GEQU  $04 ; Pointer - QDProcs -
ostdRect      GEQU  $08 ; Pointer - QDProcs -
ostdRRect     GEQU  $0C ; Pointer - QDProcs -
ostdOval      GEQU  $10 ; Pointer - QDProcs -
ostdArc       GEQU  $14 ; Pointer - QDProcs -
ostdPoly      GEQU  $18 ; Pointer - QDProcs -
ostdRgn       GEQU  $1C ; Pointer - QDProcs -
ostdPixels    GEQU  $20 ; Pointer - QDProcs -
ostdComment   GEQU  $24 ; Pointer - QDProcs -
```

```
ostdTxMeas    GEQU  $28 ; Pointer - QDProcs -
ostdTxBnds    GEQU  $2C ; Pointer - QDProcs -
ostdGetPic    GEQU  $30 ; Pointer - QDProcs -
ostdPutPic    GEQU  $34 ; Pointer - QDProcs -
```

The following code fragment shows how you might replace the StdRect procedure with your own for a given window:

```
pha                      ; open a test window
pha
PushLong #MWindData      ; standard setup for NewWindow
_NewWindow
_SetPort

PushLong #MyProcs         ; get a record to modify
_SetStdProcs

ldy #ostdRect             ; get the low word of my rectangle routine
lda #myRect-1              ; (minus one) and patch it in to the record
sta myProcs,y
lda #^myRect               ; do the same for the high word
sta myProcs+2,y

PushLong #MyProcs          ; install the procs
_SetGrafProcs
```

The interface to bottleneck procedures is different from the interface to other QuickDraw II routines; you do not make calls via the tool dispatcher and you pass most parameters on the direct page and in registers (rather than on the stack). To write your own bottleneck procedures, you have to know where the inputs to each call are kept and how to call the standard procedures from inside your own procedures.

The standard bottleneck procedures are accessed through vectors in bank \$E0.

```
StdText      $E01E04
StdLine      $E01E08
StdRect      $E01E0C
StdRRect     $E01E10
StdOval      $E01E14
StdArc       $E01E18
StdPoly      $E01E1C
StdRgn       $E01E20
StdPixels    $E01E24
StdComment   $E01E28
StdTxMeas   $E01E2C
StdTxBnds   $E01E30
StdGetPic   $E01E34
StdPutPic   $E01E38
```

When you call any of the standard procedures, the first direct page of QuickDraw II is active. If you pass variables on any direct page other than the first (direct page locations greater than \$FF), you can use a simple trick to access them. For example, to access TheFillPat (\$10E) without changing the direct page register:

```
idx      #$100           ;offset to second DP
lda      >$0E,X           ;gets "DP" location $10E
```

Certain locations on the direct page are always valid:

PortRef	\$24
MaxWidth	\$28
MasterSCB	\$08
UserID	\$0A

DrawVerb is usually valid, but not always:

**DrawVerb** \$38

Each of the bottleneck procedures uses the direct page differently.

QuickDraw II has an interesting bug relating to the standard conic bottleneck procedures. If you replace any of the standard procedures with your own, QuickDraw II does not perform some of the setups it normally would before calling the standard conic procedures (**stdRRect**, **stdOval**, **stdArc**). For example, if you replace StdRect with a custom rectangle routine, but leave the other conic pointers alone (as shown in the code fragment above), QuickDraw II will not do all of the normal setups when calling the standard conic routines. To deal with this bug of QuickDraw II, you must patch out the additional bottleneck procedures and set up those direct pages locations yourself, or the results will not be what you expect. The QuickDraw II direct-page variables you must initialize yourself in this instance are bulleted (•) below.

#### **StdText**

<b>DrawVerb</b>	\$38	Describes the kind of text to draw. There are three possible values:  DrawCharVerb      0 DrawTextVerb      1 DrawCStrVerb      2
<b>TextPtr</b>	\$DA	If the draw verb is DrawTextVerb or DrawCStrVerb, <b>TextPtr</b> points to the text buffer or C string to draw.
<b>TextLength</b>	\$D8	If the draw verb is DrawTextVerb, <b>TextLength</b> contains the number of bytes in the text buffer.
<b>CharToDraw</b>	\$D6	If the draw verb is DrawCharVerb, <b>CharToDraw</b> contains the character to draw.

#### **StdLine**

<b>Y1</b>	\$A6	Starting Y value for the line to draw
<b>X1</b>	\$A8	Starting X value for the line to draw
<b>Y2</b>	\$AA	Ending Y value for the line to draw
<b>X2</b>	\$AB	Ending X value for the line to draw
<b>Rect2</b>	\$AE	Exactly the same thing as Y1, X1, Y2 and X2 in the top, left, bottom, and right of the rectangle

#### **StdRect**

<b>DrawVerb</b>	\$38	One of the following five drawing verbs:  Frame      0 Paint      1 Erase      2 Invert      3 Fill      4
<b>Rect1</b>	\$A6	The rectangle to draw in standard form (top, left, bottom, right)

TheFillPat \$10E The pattern to use for the rectangle if the verb is Fill

**Note:** The QuickDraw II Auxiliary `SpecialRect` call does not use the rectangle bottleneck procedures.

**StdRRect**

DrawVerb	\$38	One of the following five drawing verbs: Frame 0 Paint 1 Erase 2 Invert 3 Fill 4
Rect1	\$A6	The boundary rectangle for the round rectangle
OvalRect	\$295	A copy of the boundary rectangle for the round rectangle
OvalHeight	\$208	The oval height for the rounded part of the round rectangle
OvalWidth	\$20A	The oval width for the rounded part of the round rectangle
• ArcAngle	\$D2	Must be 360
• StartAngle	\$D4	Must be zero
TheFillPat	\$10E	The pattern to use for the round rectangle if the verb is Fill

**StdOval**

DrawVerb	\$38	One of the following five drawing verbs: Frame 0 Paint 1 Erase 2 Invert 3 Fill 4
Rect1	\$A6	The boundary rectangle for the oval
OvalRect	\$295	A copy of the boundary rectangle for the oval
• OvalHeight	\$208	Must be the height of the oval
• OvalWidth	\$20A	Must be the width of the oval
• ArcAngle	\$D2	Must be 360
• StartAngle	\$D4	Must be zero
TheFillPat	\$10E	The pattern to use for the oval if the verb is Fill

**StdArc**

DrawVerb	\$38	One of the following five drawing verbs: Frame 0 Paint 1 Erase 2 Invert 3 Fill 4
Rect1	\$A6	The boundary rectangle for the arc
• OvalWidth	\$20A	Must be the width of the boundary rectangle for the arc
ArcAngle	\$D2	The number of degrees the arc will sweep
StartAngle	\$D4	The starting position of the arc
TheFillPat	\$10E	The pattern to use for the arc if the verb is Fill



**StdPoly**

DrawVerb	\$38	One of the following five drawing verbs: Frame 0 Paint 1 Erase 2 Invert 3 Fill 4
RgnHandleA	\$50	The handle to the polygon data structure
TheFillPat	\$10E	The pattern to use for the polygon if the verb is Fill

**StdRgn**

DrawVerb	\$38	One of the following five drawing verbs: Frame 0 Paint 1 Erase 2 Invert 3 Fill 4
RgnHandleC	\$70	The handle to the region to draw
TheFillPat	\$10E	The pattern to use for the region if the verb is Fill

**StdPixels**

SrcLocInfo	\$CC	The LocInfo record for the source pixel map
DestLocInfo	\$0C	The LocInfo record for the destination pixel map
SrcRect	\$DC	The source rectangle for the operation in local coordinates for the source pixel map (as described in the source LocInfo record)
DestRect	\$1C	The destination rectangle for the operation in local coordinates for the destination pixel map (as described in the destination LocInfo record)
XferMode	\$E4	The mode to use for data transfer
RgnHandleA	\$50	The handle to the first region to which drawing is clipped (usually the ClipRgn from the GrafPort) A NIL handle is not allowed. To signify no clipping, pass a handle to the WideOpen region, which is defined as 10 bytes:  Length \$A (word) -MaxInt -\$3FFF (word) -MaxInt -\$3FFF (word) +MaxInt +\$3FFF (word) +MaxInt +\$3FFF (word)

RgnHandleB	\$60	The handle to the second region to which drawing is clipped (usually the VisRgn from the GrafPort) A NIL handle is not allowed. To signify no clipping, pass a handle to the WideOpen region.
------------	------	---

RgnHandleC	\$70	The handle to the second region to which drawing is clipped (usually the mask region from the <b>CopyPixels</b> or the <b>PaintPixels</b> call) A NIL handle is not allowed. To signify no clipping, pass a handle to the <b>WideOpen</b> region.
------------	------	---

**StdComment**

TheKind	\$A6	The kind of input for the comment
TheSize	\$A8	The number of bytes to put into the picture
TheHandle	\$AA	The data to put into the picture

**StdTxMeas**

DrawVerb	\$38	Describes the kind of text to draw. There are three possible values: DrawCharVerb 0 DrawTextVerb 1 DrawCStrVerb 2
TextPtr	\$DA	If the draw verb is DrawTextVerb or DrawCStrVerb, TextPtr points to the text buffer or C string to draw.
TextLength	\$D8	If the draw verb is DrawTextVerb, TextLength contains the number of bytes in the text buffer.
CharToDraw	\$D6	If the draw verb is DrawCharVerb, CharToDraw contains the character to measure.
TheWidth	\$DE	The resulting width should be put here.

**StdTxBnds**

DrawVerb	\$38	Describes the kind of text to draw. There are three possible values: DrawCharVerb 0 DrawTextVerb 1 DrawCStrVerb 2
TextPtr	\$DA	If the draw verb is DrawTextVerb or DrawCStrVerb, TextPtr points to the text buffer or C string to draw.
TextLength	\$D8	If the draw verb is DrawTextVerb, TextLength contains the number of bytes in the text buffer.
CharToDraw	\$D6	If the draw verb is DrawCharVerb, CharToDraw contains the character to draw.
RectPtr	\$D2	Indicates the address to put the resulting rectangle.

**StdGetPic**

This call takes input on the stack rather than the direct page. This is the one standard bottleneck procedure which you call with the direct page register set to something other than the direct page of QuickDraw II; it is set to a part of the stack.

**Stack Diagram on Entrance to StdGetPic**

Previous Contents	
DataPtr	Pointer to destination buffer
Count	Integer (unsigned) (bytes to read)
RTL Address	3 bytes

----- Top of Stack

Stack Diagram just before exit from **StdGetPic**  
Previous Contents  
RTL Address 3 bytes  
----- Top of Stack

### **StdPutPic**

This call takes input on the stack rather than the direct page; however, unlike **StdGetPic**, the direct page for QuickDraw II is active when you call this routine.

Stack Diagram on Entrance to **StdPutPic**

Previous Contents  
DataPtr Pointer to source buffer  
Count Integer (unsigned) (bytes to read)  
RTL Address 3 bytes  
----- Top of Stack

Stack Diagram just before exit from **StdPutPic**

Previous Contents  
RTL Address 3 bytes  
----- Top of Stack

## **Dealing with the Cursor**

The cursor can get in your way when you want to draw directly to the screen. QuickDraw II has two low-level routines which help you avoid this problem: **ShieldCursor** and **UnshieldCursor**. **ShieldCursor** tells QuickDraw II to hide the cursor if it intersects the **MinRect** and to prevent the cursor from moving until you call **UnshieldCursor**.

There is a bug in **ShieldCursor** for System Disks 4.0 and earlier. This bug is related to the routine **ObscureCursor**. When the cursor is obscured, **ShieldCursor** does not prevent the cursor from moving; therefore, the user is able to move the cursor during a QuickDraw II operation, and this movement may disturb the screen image.

Calls to **ShieldCursor** **must** be balanced by calls to **UnshieldCursor**. You may not call **ShieldCursor** successively without calling **UnshieldCursor** after each call to **ShieldCursor**. There is no error checking, so careless use of these routines will result in an unusable system.

**MinRect** is the smallest possible rectangle which encloses all the pixels that may be affected by a drawing call. You keep **MinRect** on the direct page and usually calculate it by intersecting the rectangle of the object you are drawing with the **BoundsRect**, **PortRect**, boundary box of the **VisRgn**, and the boundary box of the **ClipRgn**. You must set up **MinRect** yourself.

**ShieldCursor** also looks at two other fields on the direct page of QuickDraw II. **ImageRef** is a long word located at \$0E. If **ImageRef** does not point to \$E12000 or \$012000, QuickDraw II assumes you are not drawing to the screen, so it does not have to shield the cursor. **BoundsRect** is a rectangle located at \$14, and QuickDraw II uses it to translate **MinRect** into global coordinates. These values are generally correct, but under the following known circumstance, they are not and **ShieldCursor** does not function properly:

1. You have just drawn to an off-screen GrafPort with QuickDraw II.
2. You switch to a GrafPort on the screen.
3. You call **ShieldCursor**.

**ImageRef** and **BoundsRect** are not updated until QuickDraw II is actually committed to drawing, thus, these values are still for the off-screen GrafPort in this case, even though you switched to a GrafPort on the screen. Therefore, when you call **ShieldCursor**, you have to make sure that these values are current. (If these values are current, **ShieldCursor** will work correctly, no matter what the circumstances.)

You can find the location of the QuickDraw II direct page with the **GetWAP** call. For speed reasons, you may not want to make the **GetWAP** call for each **ShieldCursor** call. You may wish to get the work area pointer value after starting QuickDraw II and store it for future reference.

#### Calling **ShieldCursor**:

1. Set direct page for QuickDraw II.
2. Save the existing values of **MinRect**, **ImageRef**, and **BoundsRect**.
3. Set **MinRect**, **ImageRef**, and **BoundsRect**.
4. Let QuickDraw II know you've changed the contents of its direct page by clearing the "dirty" flags bits 14 to 0:

```
DirtyFlags    equ     $EC

        ldx      #$200          ;index to QD's third page of work space
        lda      DirtyFlags,x
        and      #$8000
        sta      DirtyFlags,x
```

5. JSL to **ShieldCursor**.
6. Restore the previous values of **MinRect**, **ImageRef**, and **BoundsRect**.

**Note:** Saving and restoring these values was not previously mentioned in this Note and in most circumstances it is not necessary. Saving and restoring is now recommended. In particular, if **ShieldCursor** is called inside a QuickDraw II bottleneck procedure, the system can crash if you fail to restore the contents of direct page.

#### Calling **UnshieldCursor**:

1. Set direct page for QuickDraw II.

|2. JSL to UnshieldCursor.

<b>ShieldCursor</b>	<b>\$E01E98</b>
MinRect	\$00
ImageRef	\$0E
BoundsRect	\$14

<b>UnshieldCursor</b>	<b>\$E01E9C</b>
-----------------------	-----------------

---

#### Further Reference

- *Apple IIgs Toolbox Reference, Volume 2*